AFRL-SR-AR-TR-10-0201

REPORT DOCUMENTATION PAGE

PLEASE DO NOT RETURN YOUR FORM TO THE ABOVE ORGANIZATION.

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
05-02-2010	Final Technical Report	01-05-2008 - 31-10-2009

4. TITLE AND SUBTITLE
"Process Integrated Mechanism for Human Computer Collaboration and Coordination"

Novel Approaches to Human-Computer Collaboration and Coordination

5a. CONTRACT NUMBER

5b. GRANT NUMBER
FA9550-08-1-0218

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Dr. James Allen

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Florida Institute for Human & Machine Cognition (IHMC)
40 South Alcaniz Street
Pensacola, Florida 32502

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)

10. SPONSOR/MONITOR'S ACRONYM(S)

11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unrestricted

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The PIM model addresses the problems associated with traditional approaches such as centralized, swarm based, or multi-agent based approaches. Compared to centralized approaches, it ameliorates the robustness problem because no single node has to be in control. The PIM approach also ameliorates the communication bottleneck by moving the processing to the data rather than the data to the process. The PIM approach retains the conceptual simplicity and ease of programming of the centralized model as it removes the complications of negotiation protocols and timing problems in coordinating multiple autonomous systems.

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			Larry Warrenfeltz
U	U	U	Unlimited	10	19b. TELEPHONE NUMBER *(Include area code)* 850-202-4473

Final Report:
Novel Approaches to Human-Computer Collaboration and Coordination

Background

Many military and civilian scenarios ranging from combat operations to surveillance and reconnaissance to search and rescue can benefit significantly from teams of humans, robots, unmanned vehicles, and other computers that collaborate and coordinate together to solve the problem at hand. Efficient coordination mechanisms can greatly improve safety, robustness, quality and efficiency of the overall human-machine operation, but designing and building systems of humans and computers that coordinate well is a challenging problem, both in terms of the complexity of developing and verifying the algorithms as well as the runtime efficiency.

This project addressed this challenging problem by developing and applying a novel approach called a Process Integrated Mechanism (PIM). We developed the core PIM model, developed prototype implementations, and demonstrated the viability of approach in a series of demonstrations.

Prior research on coordination and collaboration has focused on both centralized and distributed approaches, but both approaches have inherent limitations. The centralized approach requires a coordinator to operate at a single point in the system, and this coordinator needs to have complete information about all the entities involved. The resulting negative implications are that the entire state of the system has to be transferred and collected at a single point and that a single-point failure of the coordinator will cause the overall system to fail.

Several distributed approaches have been proposed and explored: process coordination systems such as Linda, multi-agent approaches, and emergent behavior approaches inspired by biological systems such as swarms of bees, colonies of ants, schools of fish or flocks of birds. These distributed approaches do not have a single point of failure but significantly increase the complexity of the programming model and the runtime. The increase in complexity is caused by the requirement that entities construct and maintain a model of the state and expected behavior of other entities, whether through direct communication with each other or through the environment. Moreover, the multi-agent approach and the emergent behavior approach introduce additional uncertainty in the behavior of the overall system, which is undesirable. While the emergent approaches may utilize fairly simple programming models, the behavior they induce is even more unpredictable.

Furthermore, both the centralized and distributed approaches may require significant communications bandwidth to transmit state information over the network links that interconnect the entities.

A Process Integrated Mechanism (**PIM**) is a novel programming model and a runtime architecture that addresses the distributed coordination problem. The PIM approach retains the advantage of a single coordinating authority while avoiding the structural difficulties that have traditionally led to rejection in complex settings. The components in the PIM architecture are conceived as parts of a single mechanism, even when they are physically separated and operate asynchronously. A PIM is a mechanism integrated at the *software level* rather than by physical connection. It maintains a single unified world-view, and behavior is controlled by a single coordinating process.

In this report, we briefly describe our progress in three key areas: development of a precise definition and description of the approach, development of a new runtime system that implements the PIM model, and development of demonstrations in which a PIM system is used to control a set of robots.

Result 1: Defining the Model

We completed an overview paper describing the PIM model that has been submitted for publication in *AI Magazine*. The paper describes the key ideas, presents intuitive examples, and makes the case for this radically new approach to distributed control. We present just a brief overview here so as to set the context for the rest of this report.

The core idea of the PIM is to retain the perspective of the single controlling authority but abandon the notion that it must have a fixed location in the system. Instead, the computational state of the coordinating process is rapidly moved among the component parts of the PIM. More precisely, a PIM consists of a single *Coordinating Process* (CP) and a set of *Components* each capable of running the CP. The CP cycles amongst the components at a speed that is sufficient to meet the overall coordination needs of the PIM, so that it can react to new events in a timely manner. The time that the CP runs on a component is called its *Residency Time*. Each component maintains the code for the CP, so the controlling process can move from component to component by passing only a small run-time state using the mechanisms of *strong mobility* (Suri et al, 2000). The underlying PIM runtime system manages the actual movement of the CP across the components, and presents the programmer with a virtual machine in which there is a single coordinating process operating with a unified global view although, in fact, the data and computation remains distributed across the components.

The PIM model addresses some key problems associated with traditional approaches. In comparison to centralized approaches, the PIM model ameliorates the robustness problem because the coordinating process is not resident (except transitorily, with backups at all other nodes) at any one location. It also ameliorates the communication bottleneck by moving the process to the data rather than the data to the process. While multi-agent systems address the robustness issue, they introduce significant complexity and uncertainty in achieving highly coordinated behavior. The PIM approach removes the complications of negotiation protocols and timing problems in distributed systems, enabling an ease of programming and conceptual simplicity similar to centralized models, and often offering conceptual advantages over the centralized model.

The PIM model presents the programmer with an intuitive abstraction that greatly simplifies controlling distribution coordination. One can draw an analogy to Time-sharing. Time sharing models revolutionized computing because they allowed multiple processes to run on the same computer at the same time as though each was the only process running on that machine. The programmer could construct the program with no concern for the details of the process switching that is actually happening on the processor. To the programmer it is as though their program has the entire processor, even though in reality it is only running in bursts as it is switched in and out. The PIM model, on the other hand, provides a different illusion. To the programmer the PIM appears as one process running on a single machine, but the CP is actually cycling from component to component. Furthermore, the programmer see a single uniform memory, even though memory is distributed, like in a distributed memory system. In other words, the set of

Time Sharing	*Process Integrated Mechanism*
One processor, many processes	One process, many processors
Supports the illusion that each process is the only one on the machine	Supports the illusion that there is one entity controlled by a single process
Each process sees only its 'allocated' memory (a subpart of the overall system's memory)	All memory on every processor is accessible as though it were one memory.
Processes must be switched frequently enough to maintain the illusion that each is running all the time	Process must execute on each component frequently enough to maintain illusion that it can control any of the components at any time
Programmer writes program as though it is the only one on the machine	Programmer writes program as though there is a single machine.

Table 1: Comparing the PIM model with Time-Sharing

components appear to be a single entity (i.e., a PIM). The programmer need not be concerned with the details of the process moving among the processors, moving data to the process, or even where data is actually stored.

In the time sharing model, it is important that each process is run sufficiently frequently to preserve the illusion that it is constantly running on the machine. Likewise, with the PIM model, it is important that the CP runs on each component sufficiently frequently so that each component can react appropriately to any changing circumstances in the environment. The contrast between the time-sharing model and the PIM model is shown in Table 1.

Result 2: Development of a PIM Runtime System

The first prototype implementation of the PIM runtime was developed to demonstrate the key concepts of the PIM. As part of this effort, we have significantly redesigned and re-implemented the PIM runtime to provide several new features.

The original PIM runtime suffered from several limitations. It realized only the strict round-robin migration pattern for the CP. Therefore, the CP would visit every node in the same sequence, residing for the same length of time on each node. This time-slice was configurable, but only prior to starting up the PIM runtime and launching the CP. Another limitation was that the topology of the PIM had to be pre-configured prior to starting up the PIM runtime. This caused two issues: no new nodes could be incorporated into an already operational PIM and the failure of a node would cause the PIM runtime to terminate. The latter was a significant limitation given that one of the main objectives of the PIM was to provide fault-tolerance in a transparent manner. Note that this shortcoming was purely an implementation issue, not an issue with the PIM model and approach.

The new runtime has addressed all of the above shortcomings. In particular, the new runtime provides the following new important features:

Dynamic node discovery

The new PIM runtime dynamically discovers nodes that are reachable over the network and incorporates them into the set of nodes that constitutes the PIM. If a node has not had a need to transmit any data over a fixed ping interval, the node will transmit a short ping message to notify its network neighbors regarding its presence and reachability. This allows each node in the PIM

to construct and maintain a list of neighbors. When a node sees a new neighbor for the first time, that new neighbor is incorporated into the overall PIM. Note that we optimize this behavior by only transmitting a ping when there is no other PIM-related or CP-related transmission.

Flexible CP migration

The second enhancement to the PIM runtime comes from significantly improving the migration of the CP. In particular, the CP is no longer forced to migrate in a round-robin fashion with a fixed residency time. The new runtime takes three factors into account in deciding the next node to which the CP will migrate. The first factor is the CP operation. The second factor is the network topology. The third factor is the neglect tolerance. The operation of the migration algorithm is described as follows. When the CP is executing at node n, the PIM runtime determines the next node to go to using the following steps. If the CP has executed an operation that requires that the CP be resident on a specific node, then the runtime will attempt to migrate the CP to that node next. If the specific node is not a neighbor of node n, then the PIM runtime picks the node that has not been visited for the longest time and migrates the CP to that node. If there is no specific node that the CP has to interact with, then the runtime migrates the CP to the neighboring node that has not been visited for the longest time.

This new migration algorithm also works to incorporate dynamically discovered nodes. When a new neighbor is discovered, that neighbor is inserted into the list of nodes to be visited with no prior visit time. Therefore, it will automatically move to the head of the list of nodes to migrate to and will be chosen by the runtime as the next node to migrate.

Opportunistic state recording

The PIM runtime provides fault-tolerance against node failures. Node failures may be categorized into two classes: a node failure while it is not running the CP and a node failure while it is running the CP. In the first case, the PIM runtime simply ignores the node that has failed and proceeds to migrate the CP to another node. In the second case, the PIM runtime has to recover the last known state of the CP, which is usually archived on the previous node.

In order to improve the fault-tolerance capability of the PIM, the runtime has been enhanced to allow opportunistic recording of the CP state on as many nodes as possible. In particular, a node that is transmitting the CP state always does a broadcast of the state information. This allows all nodes within the network vicinity of the transmitting node to receive and archive the CP state. This feature exploits the fact that doing a broadcast (or a multicast, which is another option) on a wireless network does not consume any more channel capacity than doing a unicast. By realizing this opportunistic state recording, the PIM runtime improves the ability of the PIM to recover the last know state of the CP significantly.

Blocking and Non-blocking Interaction

The fourth improvement to the PIM runtime has been a redesign of the API for CP development. In particular, the interaction with component parts has been improved by adding an asynchronous interaction mode. Whenever the CP wishes to execute an action on a part, the CP now has two choices – either to execute the action immediately or to schedule an action to be completed in the near future. Choosing to execute the action immediately causes the CP to block until the PIM migrates the CP to the target node. This also interacts with the flexible CP migration algorithm described earlier. On the other hand, scheduling an action to be completed in

the near future does not influence the migration and it does not cause the CP to block. In the near future, this capability will be enhanced to support specifying a tolerance interval while scheduling an action.

Other Progress

Leveraging funding from another project, we have now setup a 96-node testbed environment (see figure X below) that can also be used to test the PIM runtime. This testbed allows us to control the connectivity between nodes in the testbed, both in terms of topology as well as reliability. The testbed is also instrumented to measure all network traffic between the nodes, which will enable us to measure the efficiency of the PIM. We plan to use this testbed to further experiment with the PIM in the next year.

Figure 1: NOMADS 96-node Testbed

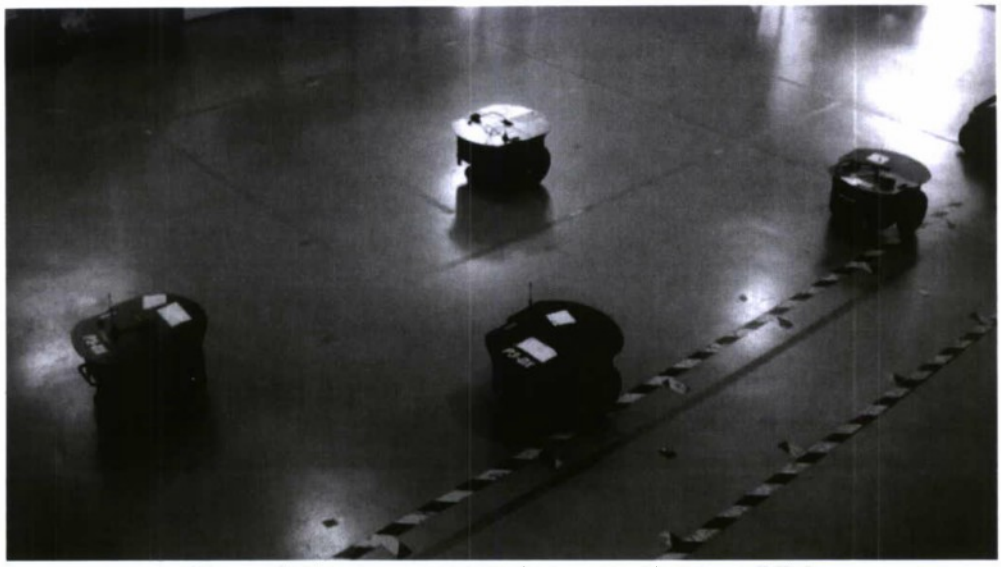


Figure 2: Pioneer robots being controlled by a PIM

Result 3: Demonstration of PIM-controlled Robot Teams

To demonstrate the viability of the PIM model, we used a PIM as a controller for a set of robots that must act in a coordinated manner. We demonstrated effective control in two different applications. The first is persistence surveillance, where a team of sensors must coordinate their activity to track a moving object while minimizing power consumption. The second is part of a pursuit scenario, where a team of robots must keep a target surrounded as it moves to try to evade capture.

Persistent Surveillance

In this scenario, we have several fixed cameras, not able to move and that capture images of a specific part of the field. We also have multiple Pioneer robots with the capability to move, each equipped with a camera. The purpose of the test is to keep track of the targets while they move in the field, minimizing the power needed to accomplish the goal.

Localization: A target in our experiment is represented by a robot that is not part of the PIM. To localize the targets we use a localization sensor on the target. The target sends information about its current location to a positioning server. The positioning server can be queried from the PIM components to know the current position of the targets. To localize the robots in the field each robot is equipped with a localization sensor that, along with odometer readings helps to compute the position of the robot in the field.

To minimize power consumption, we must turn on the cameras and transmit images only from the minimum set of cameras that allows continuous tracking of all the targets. In addition, robots should be moved only when necessary: using a fixed camera is preferred to using a robot, and turning a robot is preferred to moving a robot.

The CP: During a first phase of initialization we get the number of components of the PIM, we get the type of each component (Pioneer or Fixed Camera), we evaluate the position and heading

of each robot (for the Pioneer), and we get the area of the field each fixed camera can see (from the configuration file). At this point all is set to start a loop where:

1- we query for the position of the targets;

2- apply a set cover algorithm to decide what is the minimum set of cameras we need to turn on to keep track of all the targets and, if necessary, what robots we need to move to keep track of targets that are currently not in the field of view of any of the components;

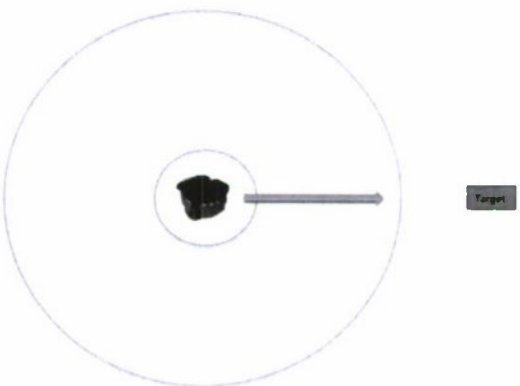
3- we move the robots and turn on and off the cameras on the components accordingly to what was evaluated.

Once we have the position of each target we apply a greedy algorithm to solve the set cover problem. The greedy algorithm evaluates how many targets each robot has in its field of view, and at each step picks the one that sees the highest number of targets, until there are no more targets to cover or no more components to accomplish the task.

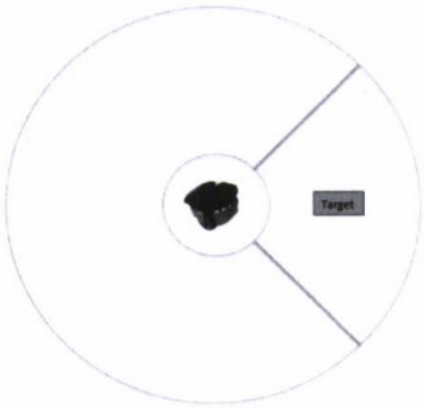
If the solution does not cover all the targets we analyze which Pioneer robot, among the ones not already selected during the set cover, requires the least power consumption to go in a position where it would have the uncovered target in its field of view. If a robot is close enough to the target, we may be able to just turn it to point to the target. Otherwise we choose the closest robot and we move it to a position where it is pointing to the target and the target is midway from the closes to the farthest point it can see.

At this point all the targets are covered, in the field of view, of one of the components of the PIM, the appropriate cameras are turned on and the images are sent to the monitoring station. The loop now cycles to check for targets movements.

How to move a robot to cover an unseen target:



In this figure we show that the target is not in the field of view of the robot. The smaller circle is the closest the robot can see and the bigger circle is the farthest the robot can see. In this case we need to move the robot to be able to have the target in its field of view, so we'll move it placing the target midway between the closest and the farthest the robot can see, between the circles in the picture, and with the robot heading the target position, as shown in the next figure.



	Target 1	Target 2	Target 3	Target 4	Target 5
Robot 1	In view				In view
Robot 2		In view	In view		
Robot 3	In view		In view	In view	
Robot 4			In view		In view

	Target 1	Target 2	Target 3	Target 4	Target 5
Robot 1	In view				In view
Robot 2		In view	In view		
Robot 3	In view		In view	In view	
Robot 4			In view		In view

Set Cover Greedy algorithm: The pictures above illustrate an example of set cover problem:

The first component selected (either Pioneer or fixed camera) will be the number 3 since it has 3 targets in its field of view. Consequently to the choice of robot 3 we eliminate from the problem the targets this robot can see: targets number 1, 3 and 4.

Now we have to cover targets 1 and 5 and none of the robots covers them both so the algorithm will select robot 2 to cover target 2 and then either robot 1 or robot 4 to cover target 5. The algorithm is concluded, no more targets need to be covered so we have a solution that is to turn on the cameras on robot 1, 2 and 3 and to turn off the camera on robot 4 (if it was previously on).

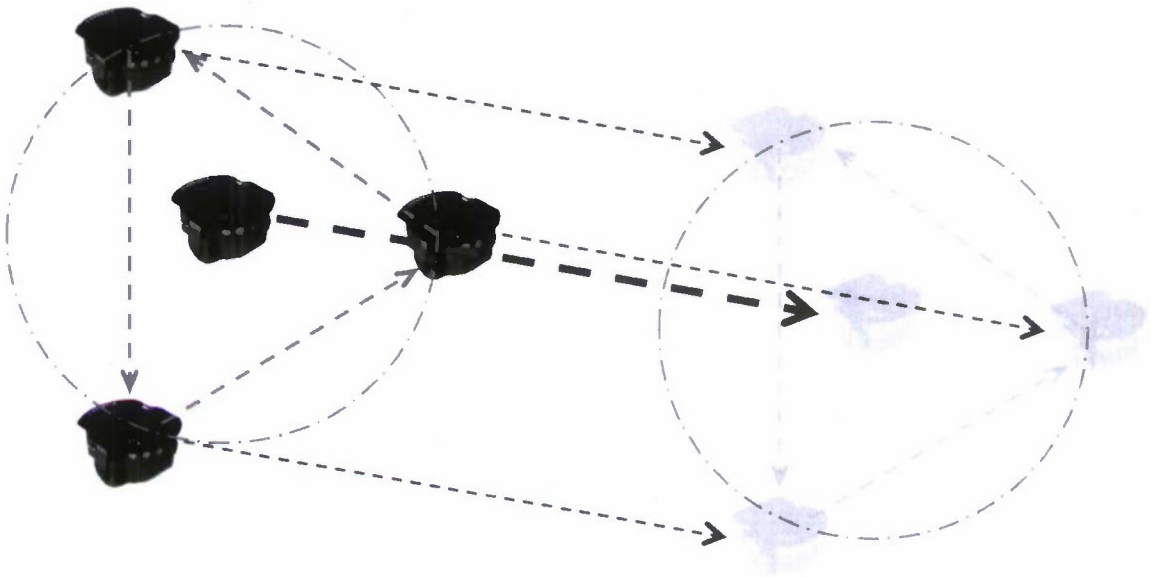


Figure 4: The guardian robots move in a coordinated fashion to keep the target robot surrounded.

5.2 Coordination in PIM

We also demonstrated coordinating robots to keep a target surrounded as it moves. The target is another robot, not part of the PIM, controlled by a human operator. The CP-controlled robots have to adapt to the moving target by repositioning to maintain their relative position from its center.

The robots controlled by the CP form a circle around the human-operated robot. The radius of the circle and the initial position of the robots are set and are translated in relation to the initial position of the human operated robot that acts as the center of the circle.

In this test we focused on keeping the guardian robots around the center robot while this moves around the field. The center robot is free to move within the area surrounded by the guardians, but if it approaches the borders of the area it will trigger a repositioning of the robots. While the overall reaction times of the robots was limited by the hardware, the PIM model displayed a robust coordination of the guardian robots, dynamically adjusting their positions to better take advantage of the current situation as the target changed directions abruptly.

We also tested the effect of the PIM residency time and overall coordination. While the robots appear to act simultaneously with CP residency times of 50ms, they become significantly less reactive at a 500ms residency time, where it takes a full 1.5 seconds to complete a CP cycle. The interesting aspect, however, was that the behavior was still reasonable despite these unrealistic delays that we imposed on the system for experimental purposes.

Summary

We made significant progress this year in defining a novel architecture for distributed control that has the robustness of distributed systems with the ease of programming of centralized systems. Besides defining the architecture, we implemented a fully capable PIM runtime system

and demonstrated its viability in a series of experiments in coordinating a set of robots. We have laid the groundwork for subsequent work focusing on demonstrating the effectiveness of the PIM model of real-world scale problems such as persistent surveillance, reconnaissance and human-robot search and rescue teams.

Publications

Sislák, D., Volf, P., Pechoucek, M., Suri, N., Nicholson, D., and Woodhouse, D. Optimization-Based Collision Avoidance for Cooperating Airplanes. In *Proceedings of the 2009 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology (WI-IAT)*, vol. 2, pp. 375-378.

Quitadamo, R., Ansaloni, D., Suri, N., Ford, K.M., Allen, J., and Cabri, G. The PIM: An Innovative Robot Coordination Model based on Java Thread Migration. In *Proceedings of the 2008 International Conference on Principles and Practice of Programming in Java (PPPJ 2008)*.

Ford, K.M., Suri, N., Kosnar, K., Jisl, P., Benda, P., Pechoucek, M., and Preucil, L. A Game-based Approach to Comparing Different Coordination Mechanisms. In *Proceedings of the 2008 IEEE Distributed Human-Machine Systems Conference (DHMS 2008)*.

Ford, K.M., Allen, J.F., Suri, N. PIM - A Novel Architecture for Coordinating Behavior of Distributed Systems, submitted to *AI Magazine*.

Sislák, D., Volf, P., Pechoucek, M., Suri, N., Nicholson, D., and Woodhouse, D. Automated Conflict Resolution Using Probability Collectives Optimizer. To Appear in *IEEE Systems, Man, and Cybernetics*.

Personnel Involved

James Allen, Principal Investigator, Research Scientist
Ken Ford, Co-PI, Research Scientist, Theoretical foundations
Niranjan Suri, Co-PI, Research Scientist, PIM Runtime
Pat Hayes, Research Scientist, Theoretical foundations
Choh Man Teng, Research Scientist, Formal analysis of recovery strategies
Maggie Breedy, Programming, Robot Interfaces
Massimiliano Marcon, Programmer, PIM Runtime
Satish Kumar, Research Scientist, Algorithms
Erika Benvegnu, Programmer, Robot Coordination
Jacques Perry, Student, programming
Matt Johnson, Programmer, Robot Control